CS 603 (Park) **Assignment VII** **Due: Mar. 31 (Mon.), 1997**

PROBLEM 1

Submit a 1–2 page project proposal. Start with a paragraph that outlines the basic idea, goal, and nature of the project. Clearly state what problem is being addressed, why it is relevant, and what you consider to be your planned contribution. In the remainder, give a more detailed account of your approach and expected outcome. Make sure to focus the scope of the project such that it can be carried out within the given time frame.

PROBLEM 2

(a) The *group membership problem* consists of The *Byzantine generals* problem consists of n generals who must, through message exchanges, agree upon a common battle plan. Some generals are traitors and their goal is to prevent the nontraitors ("good guys") from reaching agreement by impersonation and other malicious activities. Authentication is not available and up to k generals can engage in treason where $n \geq 3k+1$. Design a consensus protocol whereby the nontraitors reach agreement given a predictable (i.e., bounded delay) communication environment. First, sketch the idea and logical structure underlying your approach. Give a careful argument of the correctness of your algorithm.

(b) Does the problem become easier if authentication is available? If so, how, and sketch an algorithm. Indicate if your protocol can tolerate a larger number of faults k (relative to n) as a result of this relaxation.

PROBLEM 3

Implement the greedy load balancing algorithm discussed in class using 8 Xinu machines given the following set-up. On each host, tasks arrive with an exponentially distributed inter-task arrival time, with mean inter-arrival time $1/\lambda = 3$ sec. Confer the function `exponential_dist` with parameter λ to generate the arrival process. For fractional numbers, round them up to the nearest second.

In addition to the arrival time, we also associate to each task z_i its weight w_i which represents its CPU requirement. The CPU of all hosts are assumed to be homogeneous delivering processing performance at 10 *weight units* (an artificial unit) per second. Assume w_i is uniformly distributed over the domain $1 \leq w_i \leq 50$. Task progress is updated every second using a round robin schedule at granularity of 1 weight unit until all 10 weight units are expended. If the combined sum of the weights of the tasks at a host exceed 10, then the ID of the last task serviced is remembered and, at the next second, service is resumed beginning at the following task. A task is assumed to have terminated if its weight has been decremented to 0. The *load* at a host at some time t is defined to be the number of nonterminated tasks outstanding at t.

Test your implementation, first, on a global logical connectivity pattern where all 8 Xinu machines are assumed to be completely connected. That is, the greedy algorithm when choosing to assign a newly arriving task is able to access the load information on all 8 machines and pick the minimal one. Load information is exchanged, using UDP, by periodically broadcasting one's load at 3 sec intervals (hence the global load information is an outdated approximation to the true system state). Run the test suite for 60 seconds (on all hosts) and measure the total throughput achieved over the 60 seconds at the hosts. The *throughput* at a host is defined to be the sum of the weights of all the tasks serviced within the 60 second interval.

Second, run the test suite on a local connectivity pattern where the 8 Xinu machines are organized in a ring, and access to load information is only possible to the nearest (i.e., left and right) neighbors. Use the broadcasting mechanism as before, just do not allow access to the information in the load balancing scheduler. Compare the performance of the two runs and discuss the results. Fix seeds for the pseudo-random number generator across the two runs so that the arrival process (determined by `exponential_dist`) stays invariant facilitating performance comparison.

PROBLEM 4

As a continuation of Problem 3, modify the load balancing algorithm according to the following three separate specifications yielding two separate algorithmic variants. One, decrease the load exchange period from 3 sec to 1 sec. Two, assume the weight information about a task is *known* to the load balancing scheduler. Within the greedy specifications, modify the algorithm such that this information is exploited when making placing assignments. Compare the two variants against each other and the performance of the algorithm in Problem 3 for the completely connected topology case.